An Infrastructure for the Development of Kernel Network Services. Proof of Concept: Fast UDP.

Edgar A. León
University of New Mexico

Michal Ostrowski
IBM T. J. Watson Research Center

Scientific applications demand tremendous amounts of computational capabilities. In the last few years, the computational power provided by clusters of workstations and SMPs has become popular as a cost-effective alternative to supercomputers. Nodes in these systems are interconnected using high-speed networks via "smart" Network Interface Controllers (NIC). These controllers allow the overlap of computation and communication by processing communication tasks on the NIC and computational tasks on the host processor(s).

Parallel applications running on these clusters suffer from a variety of performance and scalability problems: host processor overhead due to communication processing, data placement overhead (memory copies), overhead due to external interrupts, cost of splitting OS functionality between host and NIC, etc. These problems are caused by the poor integration of the NIC with the Operating System (OS) and applications. The interactions between these entities are complex and are key to an application's performance.

To investigate these interactions as well as to propose and study next generation NICs, we have created an infrastructure in which network interface controllers can be simulated. Simulated NICs are based on a functional model which can run arbitrary functionality and may interact with the host in novel ways such as injecting data directly into a processor's cache. As a proof of concept, we extended the implementation of a simple unreliable communication protocol, UDP, by applying three network optimizations. This new implementation of UDP, which we have called *Fast UDP*, provides significant performance advantages over traditional UDP.

Our network infrastructure has been implemented in *Mambo*, the IBM PowerPC full-system simulator, through a *shim layer*. This layer allows the creation and dynamic loading of simulated network devices into the Mambo system simulator. Simulated NICs can be written in *C* and can be developed without Mambo source code. These controllers interact with Mambo through a well-defined interface which provides a functional abstraction of the NIC's hardware/firmware. By explicitly defining this API, NIC independent code can be developed and potentially run in any NIC that implements this abstraction. The shim interface provides control and data paths to the host, as well as functions to inject data into a processor's cache directly. Our simulated NIC is accessed by the host through memory mapped registers. Using this mechanism, a user process can interact with the NIC directly, allowing *OS and Hypervisor bypass*. Access to these registers is controlled by the OS and/or Hypervisor.

Fast UDP, a high-performance implementation of UDP, is divided into code running on the host and code running on the NIC. The code running on the host is the Linux UDP/IP stack unmodified. The code running on the NIC implements three network optimizations: message matching on the NIC; splintering data and control information from network packets; and NIC offloading.

In commodity UDP implementations, when a packet arrives from the network, the NIC copies the message to a kernel buffer and raises an interrupt for the host OS to handle this packet. The OS processes the packet through the UDP/IP stack and finally copies the payload to user space. In our approach, the NIC has been instrumented to partially process UDP packets so that the payload is transfered directly from the NIC to user space, while the header (control information) is copied to a kernel buffer. Thus the kernel remains aware of incoming network packets but does not incur in the overhead of processing application's data (including an extra copy to user space).

Message matching semantics of UDP are based on an IP address and a Port. Message matching in Fast UDP is also performed on the NIC. The information about receive UDP buffers is shared by the OS with the NIC when a user posts a UDP receive. When a UDP packet arrives from the network, the NIC matches the packet using its destination port, and if a user has posted a receive for that port, the payload will be delivered to the user buffer. UDP checksum on the packet is performed (offloaded) on the NIC to avoid the transfer of erroneous data to the user.

To compare a traditional UDP implementation with Fast UDP, we created a simple UDP application in *K42* (a high-performance research OS) and measured its runtime. The application consists of two phases: the reception of a number of packets; and performing computation on the data. Fast UDP performed 5% better than UDP when the application spends 80% of its time computing. This performance improvement is expected to increase as the application's communication to computation ratio increases.

In conclusion, we have created an infrastructure to simulate network interfaces which allow us to: (a) Better understand recent and future network architectures to fully take advantage of their capabilities; (b) Make a case for optimizations that improve application performance and scalability and provide arguments for those ones who do not; (c) Better understand the interactions between the Operating System, Applications and smart NICs to avoid bottlenecks in the data path from the network all the way to the application. As a proof of concept, we applied three network optimizations techniques to improve application performance: matching on the NIC, NIC offloading, and splintering of control and data. We obtained significant performance improvements even for a computation-bound application.